

# McStas advanced language features

- and other important, not so well known details



†Peter Willendrup<sup>1,5</sup>

†Emmanuel Farhi<sup>2</sup>

†Erik Knudsen<sup>1,5</sup>

†Uwe Filges<sup>3</sup>

†Kim Lefmann<sup>4,5</sup>

<sup>1</sup>DTU Physics, Lyngby, Denmark

<sup>2</sup>Calcul Scientifique, Institut Laue-Langevin (ILL), Grenoble, France

<sup>3</sup>Niels Bohr Institute, University of Copenhagen, Copenhagen, Denmark

<sup>4</sup>Paul Scherrer Institut, Villigen, Switzerland

<sup>5</sup>ESS DMSC, Copenhagen, Denmark

# McStas



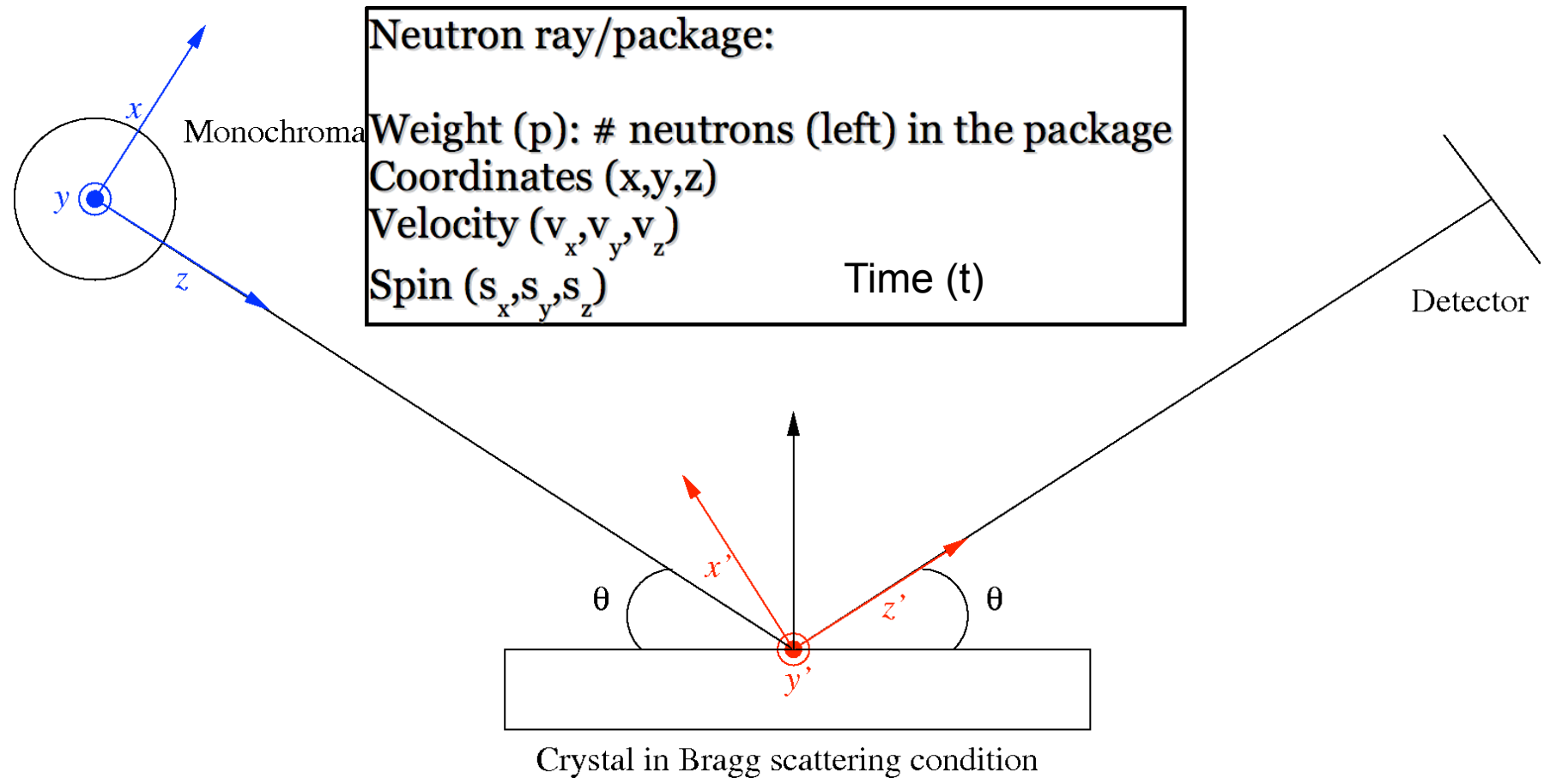
# Topics



- | A quick recap of the most important vocabulary
- | McStas language macros
- | Other important details

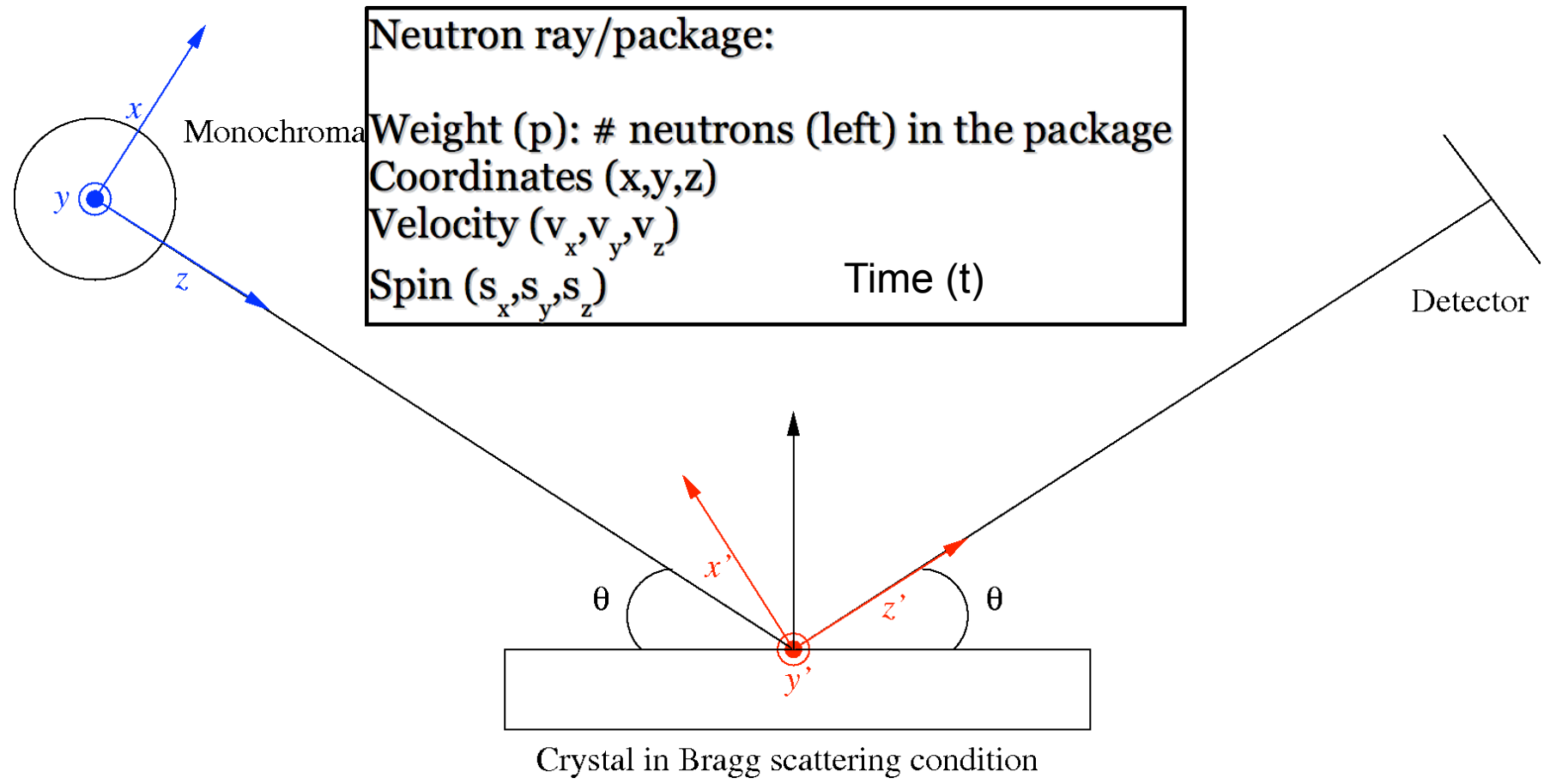


# McStas: key concepts

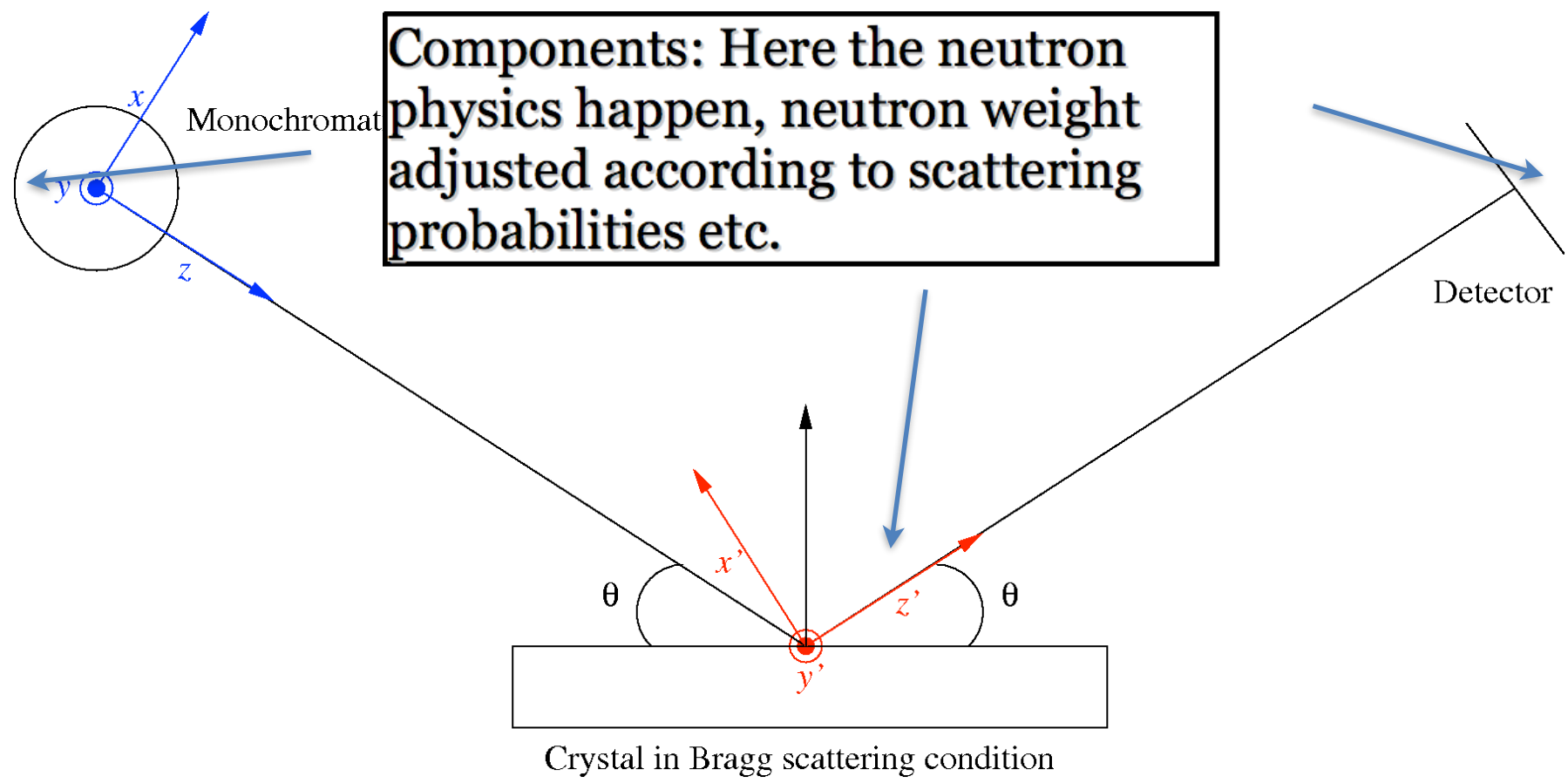


# McStas: key concepts

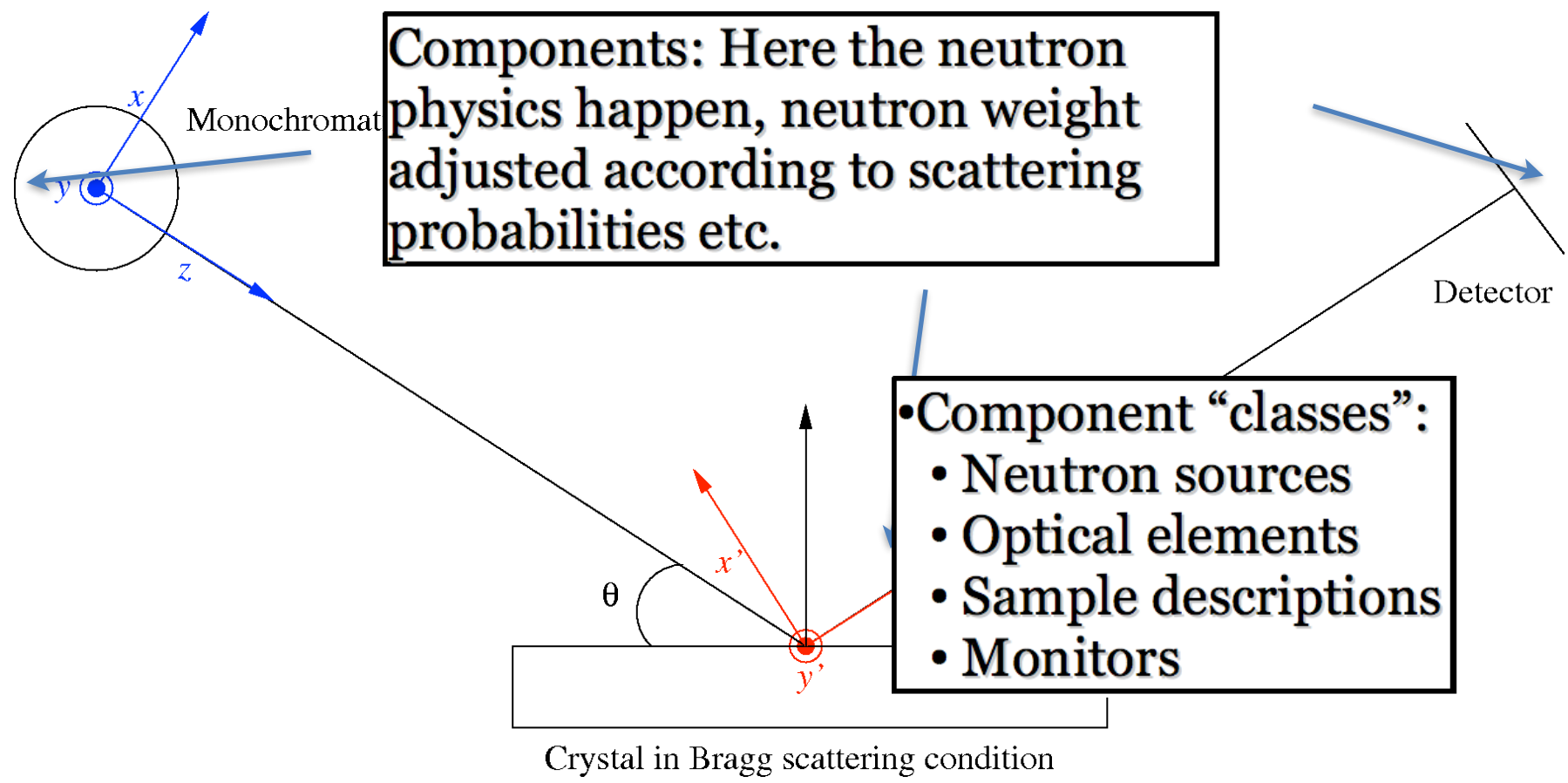
Neutron state parameters are global variables!



# McStas: key concepts

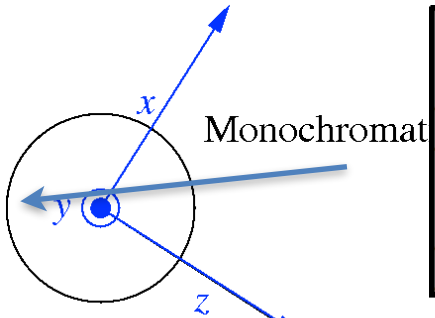


# McStas: key concepts

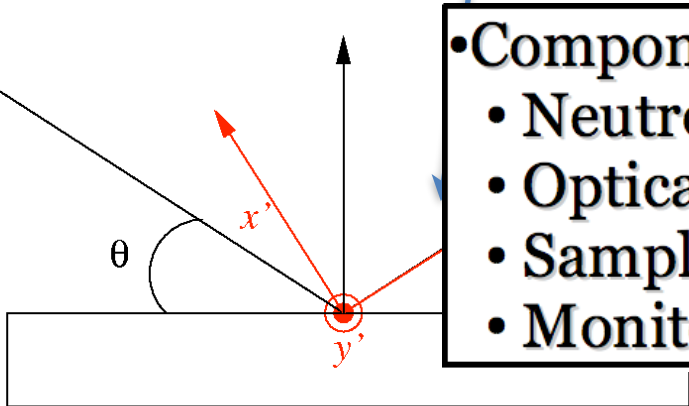


# McStas: key concepts

Local, internal coordinate system!



**Components:** Here the neutron physics happen, neutron weight adjusted according to scattering probabilities etc.

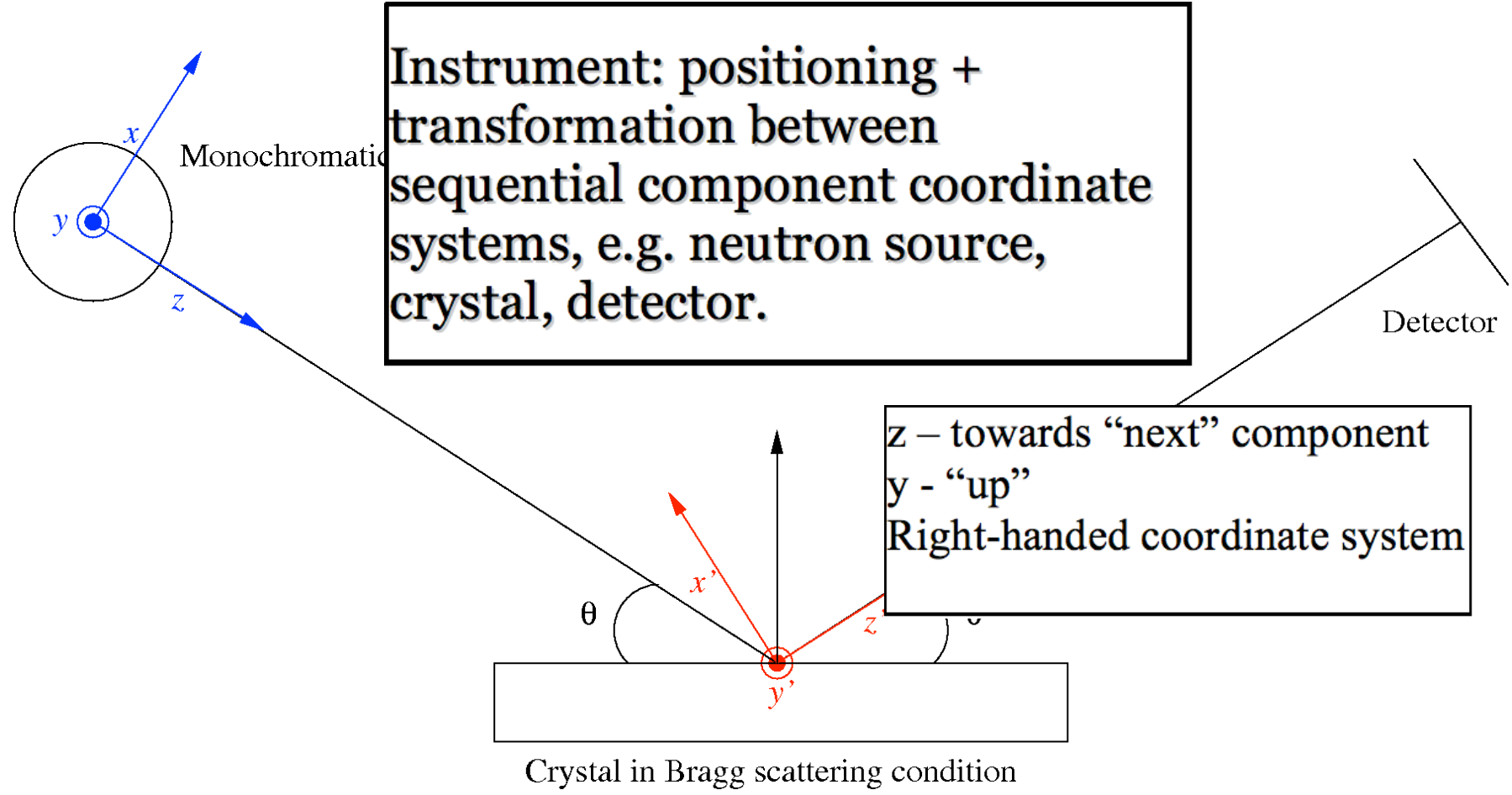


- Component "classes":
  - Neutron sources
  - Optical elements
  - Sample descriptions
  - Monitors

Crystal in Bragg scattering condition



# McStas: key concepts





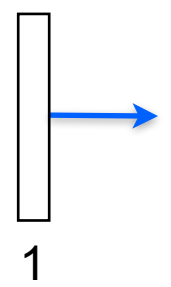
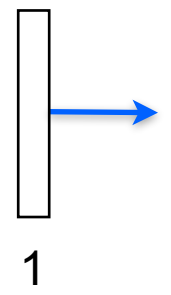
# From the “list of frequent questions”



- Why do you need to have the components in this order?



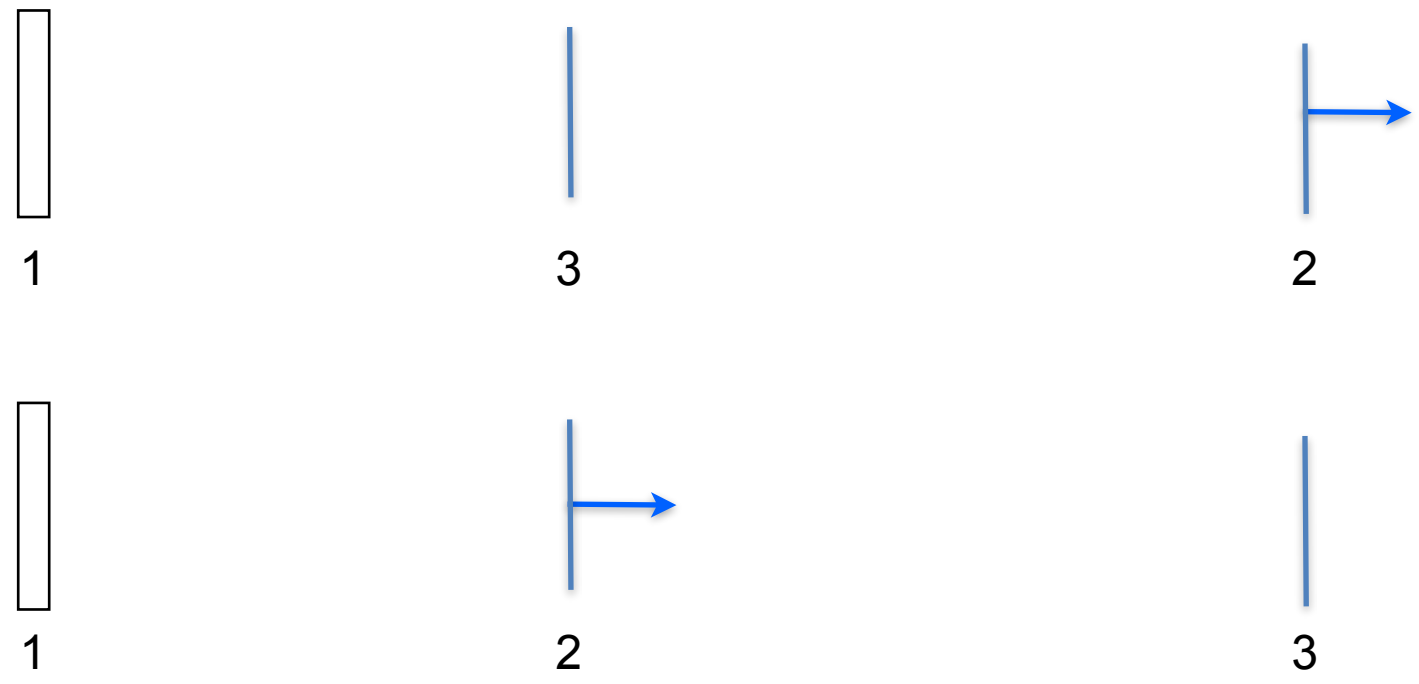
# Order of components is important



Starting at the source



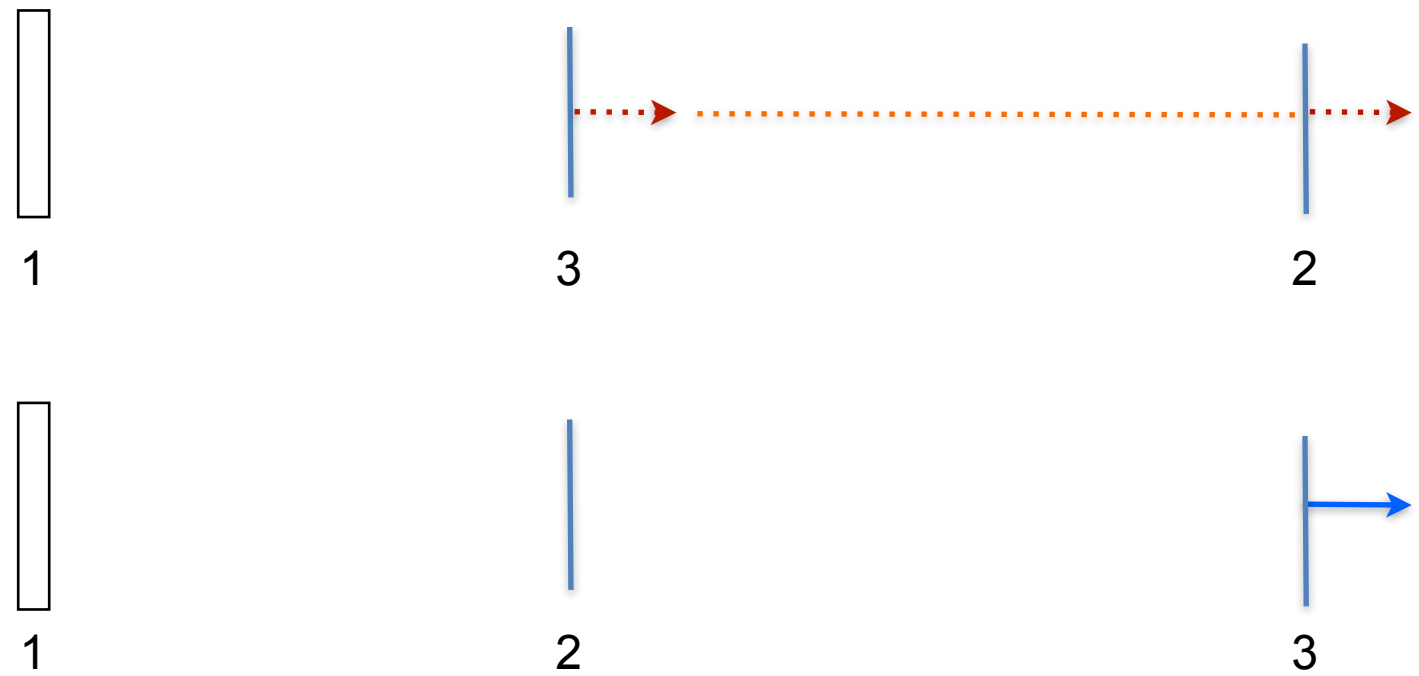
# Order of components is important



Moving to first comp in the list



# Order of components is important



Moving to 3rd comp in list requires “moving back in time”.  
Default behavior is to ABSORB this type of neutron.  
For monitors use `restore_neutron=1` in this case.  
For homegrown comps use `ALLOW_BACKPROP` macro.



# Advanced language features



▮ Macros and tricks for your instrument...



# DECLARE / INITIALIZE

- Use the DECLARE section define user variables and functions.

```
DECLARE %{  
    double myvar;  
%}
```

- Use INITIALIZE for initialization of user variables and calculations.

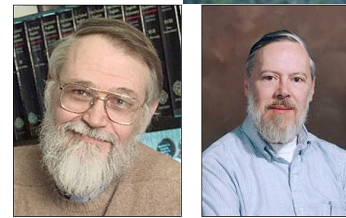
```
INITIALIZE %{  
    myvar = sqrt(PI*input_var)*rand01();  
%}
```

- Both use normal c-syntax.

- BEWARE:** (example) What you do in the c-style areas is c-standard, e.g. trigonometric functions from math.h use radians! - McStas placement specifiers work in degrees, etc...



K & R



# %include

Instrumentfiles can include external c-code or other instrumentfiles... See the examples:

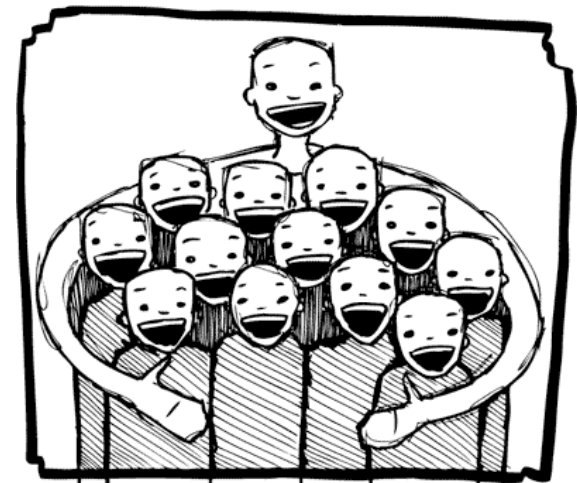
ILL\_H15\_IN6.instr:%include "monitor\_nd-lib"

ILL\_H16\_IN5.instr:%include "ILL\_H16.instr"

ILL\_H25\_IN22.instr:%include "ILL\_H25.instr"

ILL\_H25\_IN22.instr:%include  
"templateTAS.instr"

Used in the DECLARE section



# Syntax in one, complex view...

{**SPLIT**} COMPONENT name = comp(parameters) {**WHEN** condition}  
AT (...) [RELATIVE [reference|PREVIOUS] | ABSOLUTE]  
{ROTATED {RELATIVE [reference|PREVIOUS] | ABSOLUTE} }  
{**GROUP** group\_name}  
{**EXTEND** C\_code}  
{**JUMP** [reference|PREVIOUS|MYSELF|NEXT] [ITERATE number\_of\_times | **WHEN** condition] }



# SPLIT

| Increase statistics beyond this point in the instrumentfile

| SPLIT n MyArm = Arm()

| AT somewhere

| will “formulate an if-statement”:

| for j=1:n

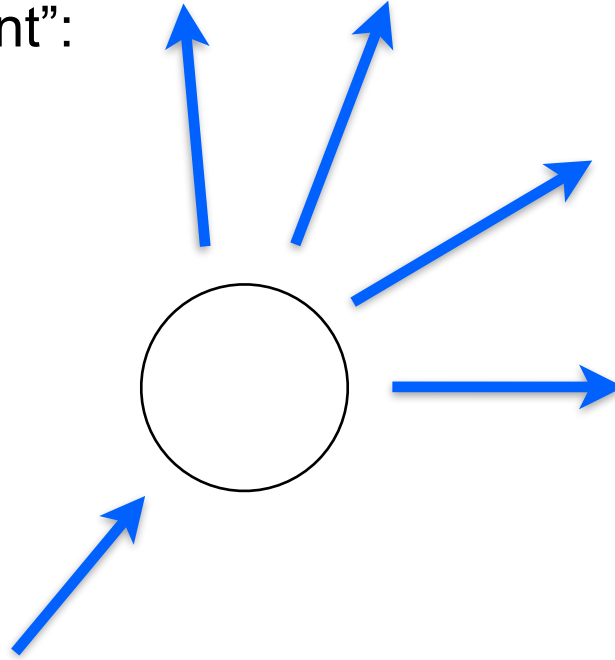
| comp1

| comp2

| comp3

| ...

| end (of instrument)



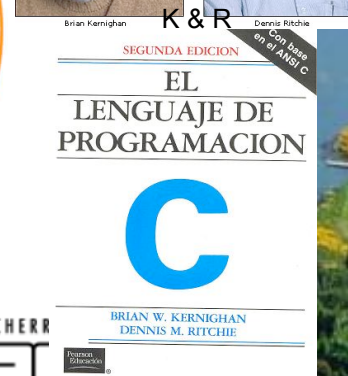
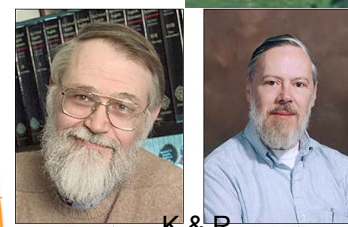
| ONLY meaningful in case of Monte

Carlo choices after SPLIT point...



# WHEN

- | Syntax:
- | COMPONENT Mine = Yours(blah, blah)
- | WHEN (c-expression) AT (....)
- | Is very powerful when combined with EXTEND and user variables, or as a method to let input parameters select if certain components are active.
- | Example: Use EXTEND to flag if neutron was scattered on one monochromator blade or another. Then later use WHEN to only show contribution from blade N at sample position?
- | COMPONENT Mon = PSD\_monitor(...)
- | WHEN (myvar==1) AT (0,0,0) RELATIVE Sample



# GROUP - components working in parallel

COMPONENT Mono1 = Monochromator\_curved(...)  
AT (0,0, -LMM) RELATIVE Cradle ROTATED (0,A1/2,0) RELATIVE Cradle  
GROUP IN6Monoks

COMPONENT Mono2 = Monochromator\_curved(...)  
AT (0,0, 0) RELATIVE Cradle ROTATED (0,A2/2,0) RELATIVE Cradle  
GROUP IN6Monoks

- One comp after the other is “tried” in sequential order until the neutron was SCATTERED.



# EXTEND



Enrich component behaviour using EXTEND:

COMPONENT Mono1 = Monochromator\_curved(...)

```
AT (0,0, -LMM) RELATIVE Cradle ROTATED (0,A1/2,0) RELATIVE Cradle
GROUP IN6Monoks
EXTEND
%{
  if (SCATTERED) { myvar = ;1 }
}%
```

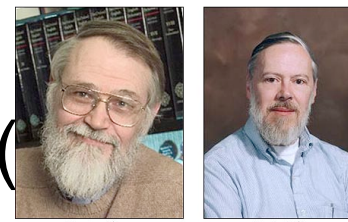


K & R

...

COMPONENT Mono2 = Monochromator\_curved(...)

```
AT (0,0, 0) RELATIVE Cradle ROTATED (0,A2/2,0) RELATIVE Cradle
GROUP IN6Monoks
%{
  if (SCATTERED) { myvar = 2 }
}%
```



Brian Kernighan

Dennis Ritchie

SEGUNDA EDICION  
EL  
LENGUAJE DE  
PROGRAMACION

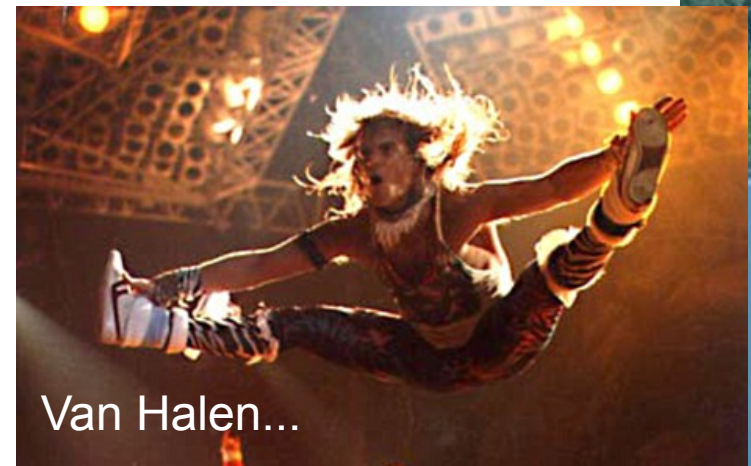
C

BRIAN W. KERNIGHAN  
DENNIS M. RITCHE



# JUMP

- | A goto. Be careful. Can be used in two situations:
  - | JUMP to myself
  - | JUMP to an Arm
  
- | No coordinate transformations are applied... (Meaning that if the Arms you JUMP between do not coincide you will “move” / “reorient” the neutrons...)
  
- | Syntaxes:
  - | COMPONENT a=b(...)
  - | WHEN (expr) AT (...) JUMP somewhere
  
  - | COMPONENT a=b(...)
  - | WHEN (expr) AT (...) JUMP somewhere



# JUMP

- | A goto. Be careful. Can be used in two situations:
- | JUMP to myself
- | JUMP to an Arm
  
- | No coordinate transformations are applied... (Meaning that if the Arms you JUMP between do not coincide you will “move” / “reorient” the neutrons...)

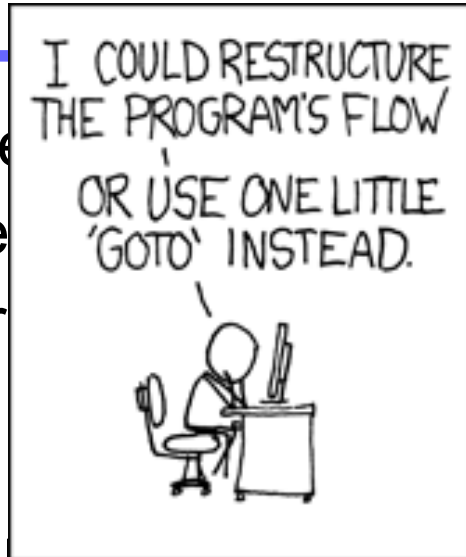
**BEWARE - This IS a GOTO!**

- | COMPONENT a=b(...)
- | WHEN (expr) AT (...) JUMP somewhere
  
- | COMPONENT a=b(...)
- | WHEN (expr) AT (...) JUMP somewhere

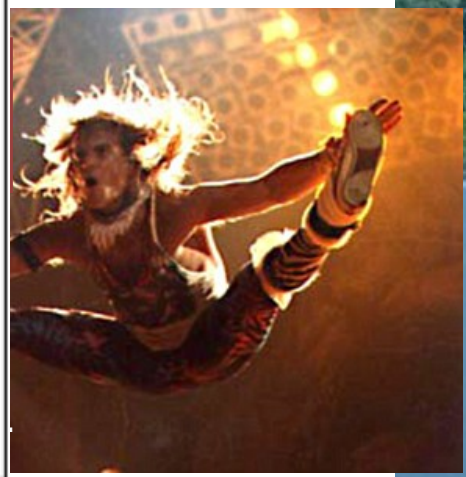
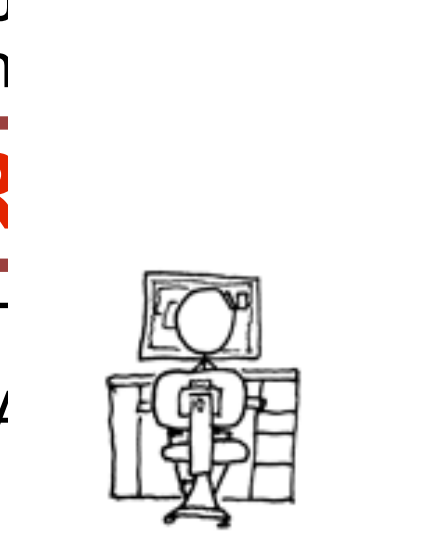


# JUMP

- | A goto. Be careful
- | JUMP to myself
- | JUMP to an Arm
- | No coordinate
- | the Arms you J
- | "reorient" the n
- | **BEWAR**
- | COMPONENT
- | WHEN (expr) /
- | COMPONENT a=b(...)
- | WHEN (expr) AT (...) JUMP somewhere



...ing that if  
will "move" /



# COPY- inside instruments



In instruments: (see ILL\_H25.instr)

COMPONENT H25\_1 = Guide\_gravity(  
w1=0.03, h1=0.2, w2=0.03, h2=0.2, l=L\_H25\_1,  
R0=gR0, Qc=gQc, alpha=gAlpha, m=m, W=gW)  
AT (0,0,Al\_Thickness+gGap) RELATIVE PREVIOUS  
ROTATED (0,Rh\_H25\_1,0) RELATIVE PREVIOUS



COMPONENT COPY(H25\_1) = COPY(H25\_1)  
AT (0,0,L\_H25\_1+gGap) RELATIVE PREVIOUS  
ROTATED (0,Rh\_H25\_1,0) RELATIVE PREVIOUS

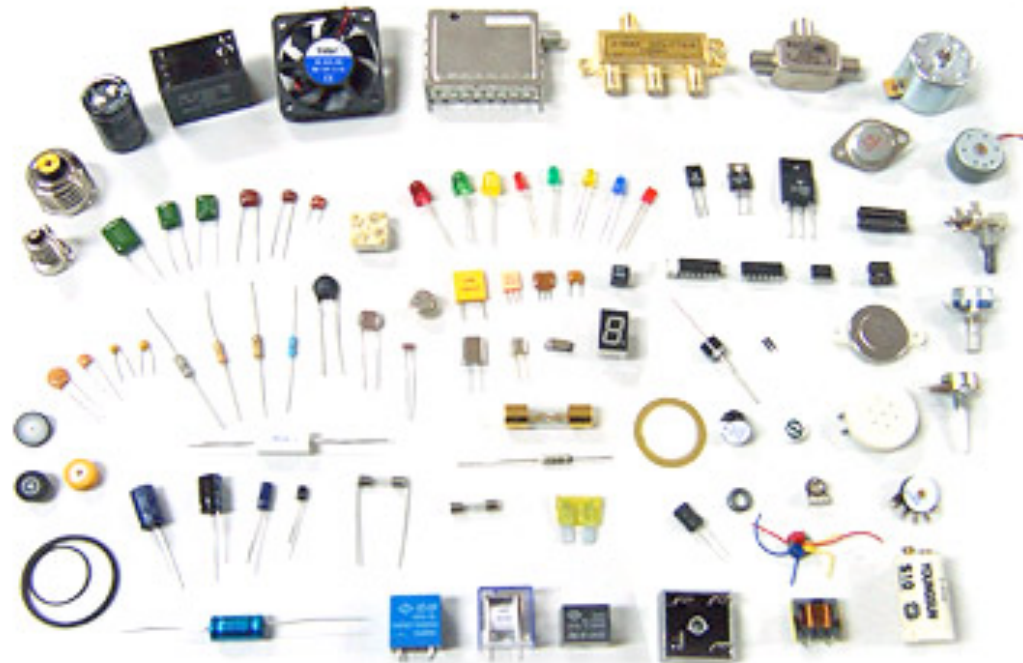


COMPONENT COPY(H25\_1) = COPY(H25\_1)(W=2\*gW)  
AT (0,0,L\_H25\_1+gGap) RELATIVE PREVIOUS  
ROTATED (0,Rh\_H25\_1,0) RELATIVE PREVIOUS



# Other advanced topics

- Tricks, macros and functions for your components



# COPY In components:



There is a heritage mechanism to create childs of existing components. These are exact duplicates of the parent component, but one may override/extend original definitions of any section.

The syntax for a full component child is

```
DEFINE COMPONENT child_name COPY parent_name
```

This single line will copy all parts of the *parent* into the *child*, except for the documentation header.

As for normal component definitions, you may add other parameters, DECLARE, TRACE, sections. Each of them will replace or extend (be catenated to, with the COPY/EXTEND words, see example below) the corresponding *parent* definition. In practice, you could y a component and only rewrite some of it, as in the following example:

```
DEFINE COMPONENT child_name COPY parent_name
```

```
SETTING PARAMETERS (newpar1, newpar2)
```

```
INITIALIZE COPY parent_name EXTEND
```

```
{
```

```
... C code to be catenated to the parent_name INITIALIZE ...
```

```
}
```

```
SAVE
```

```
{
```

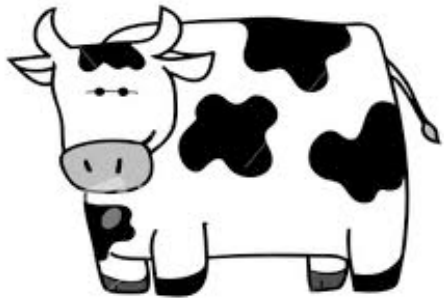
```
... C code to replace the parent_name SAVE ...
```

```
%}
```

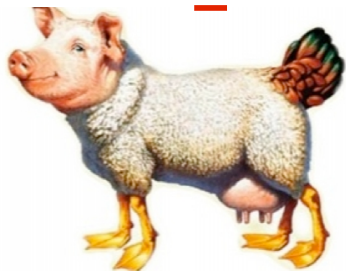
+



+



=



# Simulation statistics

- Neutron weights (intensities) in McStas are “per unit time”

- At a detector bin,  $N$  rays with weight  $p_i$  arrive.

We record:

$$I = \sum_{i=1}^N p_i$$

- Intensity

$$E = \sqrt{\frac{N}{N-1} \left| \sum p_i^2 - \left( \frac{1}{N} \sum p_i \right)^2 \right|}$$

- Statistical error, root mean square



- As noted before, McStas intensities are “per second”.
- A maximal “measurement” or integration time can be defined such that

$$I_m = \alpha_{max} I$$

$$E_m = \sqrt{I_m} \geq \alpha_{max} E$$

- Or in other words, the poisson-law based squareroot error may not become smaller than the scaled version of the statistical error. When a binned monitor is scaled, the minimal  $E_m$  for the detector bank to be used.



# The end

